

Department I - C Plus Plus

Modern and Lucid C++ for Professional Programmers

Week 1 – Introduction to C++

Prof. Peter Sommerlad / Thomas Corbat
Rapperswil, 17.09.2019
HS2019



- **Main platform is gitlab:** <https://gitlab.dev.ifs.hsr.ch/psommerl/cpp-module/>
 - It contains a repository you can clone with git
- **Each week features**
 - Lecture slides
 - Exercises (online and as PDF)
 - Solutions (except Testat and first week)
 - Wiki with additional information
- **Lecture slides and video material is available on Skripte-Server**
- **C++ Module Discord-Server**
 - Invitation link: <https://discord.gg/vrFXkEJ>

C++ Gossip

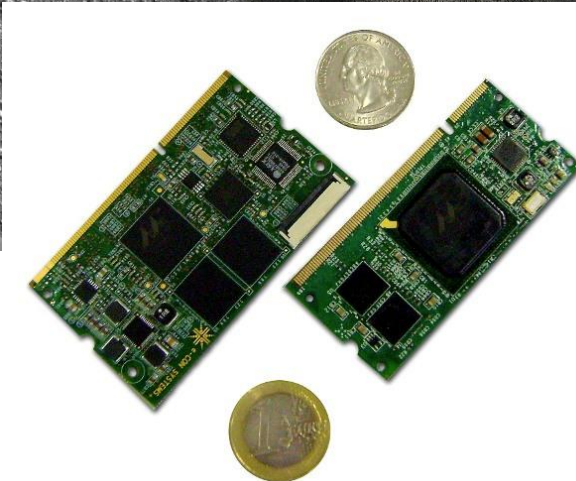
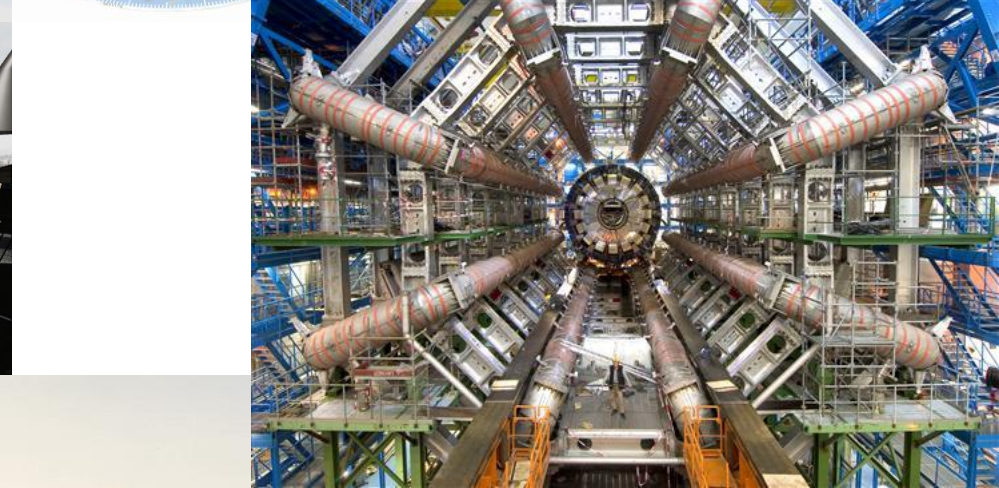


Goals:

- You know why you should learn C++
- You know about common misconceptions of C++

Why C++?

4



- **ISO standard: C++17 (C++20 forthcoming)**

- parts of it designed and decided at HSR (August 2010; June 2014; June 2018)
- works on almost all platforms: micro controller to main frame

- **Multi-paradigm language with zero-cost abstraction**

- not just object-oriented -> more mechanisms for abstraction!

- **High-level abstraction facilities**

- write less code, but need to know what actually happens!

- **Cool stuff to learn, valuable also for other languages!**



- **C++17 as of ISO 14882 (but not all: 1618 p.)**
- **Modern C++ usage, not "classic" 1990's style**
 - Much simpler, especially from 2011 ISO standard on
- **Effective use of Standard Library (STL)**
 - Avoiding re-inventing loops or containers
- **Modern Unit Testing with CUTE**
 - not having test automation is unprofessional
- **Using Cevalop (recommended)**
 - unfortunately some minor glitches, because not fully C++17 compliant yet

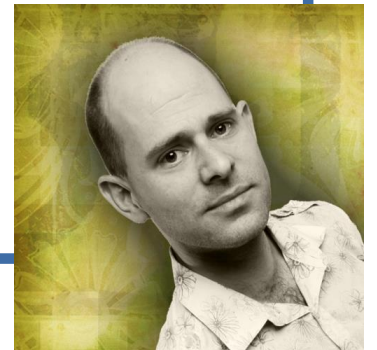


- **Well enough command of English to read**
- **Active participation:**
 - Ask questions and answer ours :-)
 - Use Discord
- **Practice, practice, practice!!!**
 - Lab tasks are obligatory (not necessarily presence)
 - Hand-ins of exercises for "Testat" in teams of two or three
- **You know about normal control structures (if, for, while)**
 - and we will teach you, how to avoid them for better abstraction!
- **You are open to learn (and unlearn) stuff**

- **Do not procrastinate! Work on the exercises right away!**
 - Catching up might be hard, because of a lot of new stuff
 - Programming is a trade and requires (a lot of) practicing
- **Plan your time for self-study during semester**
 - At least the same amount as for lectures necessary
 - And also for exercises to complete them
 - There is supplementary material on the wiki: Book chapters for most topics
- **Collaborate - form teams learning together**
 - Commitment to each other helps circumvent procrastination

*Programming in C++ is just like marriage.
It's great. It's rewarding. **But it requires work.**
You can't expect to breeze into C++ programming and for
everything to be perfect. For your code to be great with no effort.
For your life to be as easy as it was in a less committed
programming relationship.
It just doesn't pan out like that.
You have to work at it. **Put some effort in.**
But it's worth it!*

- Pete Goodliffe, CVu Feb 07 (accu.org)



- C++ is less efficient than C
- C++ is incomprehensible
- C++ requires using pointers correctly
- C++ programs crash and are a security risk
- C++ programs have memory leaks
- Plain editors like vim are best for C++ coding
- C++ code can not be unit tested

"You can't evaluate C++ from code written 10 or 20 years ago. It was a completely different language."
- Chandler Carruth, Google, 2013

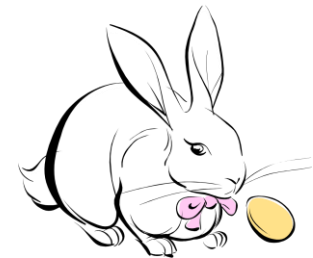
- **Compiler error messages can be hard to understand**
- **One can program C and call it C++**
- **There is no garbage collection like in Java or .NET**
 - But you don't need it!
 - if you program like we teach you
- **IDE support is not yet up to Java**
 - IFS is working on that with Cevelop
 - you can help!
- **Compilers might behave different than specified by the standard!**
- **C++ standard defines "undefined behavior"**



Undefined behavior

-- behavior, upon use of a non-portable or erroneous program construct, ... for which the standard imposes no requirements. Permissible undefined behavior ranges from ignoring the situation completely with unpredictable results, to having demons fly out of your nose."

- John F. Woods



Fixing Hello World



Goals:

- You get to know the `main()` function
- You know the basic features of Cevelop
- You can explain deficiencies in a standard Hello World program

- `main()` is the program entry function
 - The "Big Bang" in the execution of every program
- **C++ provides functions (not methods as Java)**
 - Not all functions are bound to a class or object! the latter are called "**member-functions**" not methods
- **Return types are written in front of the function name (most of the time) in declarations**
 - or `auto` if the return type should or can be deduced (main's return type must be `int`)
- **Parentheses after the name `()` mark a function**
- **Braces `{ }` mark the function body**

```
int main() {  
}
```

- **You can install Cevalop on your own environment**
- **Suggested configuration**
 - Cevalop 1.12.1 (= Eclipse CDT + IFS plug-ins)
 - Ctypechecker plug-in (Recommended, especially for testat exercises) [beta plugins]
 - g++ using -std=c++17 (GNU g++ 8.x or clang >6.0)
 - Xcode Clang might be broken (too old) on Macs
 - On MacOS use homebrew to install current g++ (8.x)
 - you might need to set PATH to include /usr/local/bin



Download IDE at:
www.cevelop.com



- Hello World program as generated by Eclipse CDT
- What's wrong?

```
//=====
// Name      : HellWorld.cpp
// Author    :
// Version   :
// Copyright  : Your copyright notice
// Description: Hello World in C++, Ansi-style
//=====

#include <iostream>
using namespace std;

int main() {
    cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
    return 0;
}
```

Belongs into a version management system

Bad practice, very bad in global scope

Ridiculous comment

Redundant

Inefficient and redundant

Using a global variable! Really bad (except in main())

Issue	Reason	Possible Solution
Header comments (Name, Author, Version) should not be part of the source code	Such information gets outdated quickly or might be plain wrong	Version management system (git) knows this information
Using directives (using namespace std;) should not be used in global scope	They pollute the namespace with the imported names	Remove using directive and use name qualifiers: std::cout and std::endl (Cevelop Refactoring)
Use of global variable cout	Global variables are hard to test, especially program in/output	Replace the global variable by a parameter (only allowed in main())
Ridiculous comment	It is obvious that this program prints !!!Hello World!!!	Delete it - if you need comments like that, do not program!
Superfluous return statement	The main() function as a unique exception to all functions does not require a return statement even though it has return type	Delete it (As long as it returns 0 in main())
Output of endl prints a new line and flushes the stream	An additional line-break is not necessary and the flush is inefficient	Remove << endl Add a line-break ('\n') to the string literal if necessary

C++ Compilation Process



Goals:

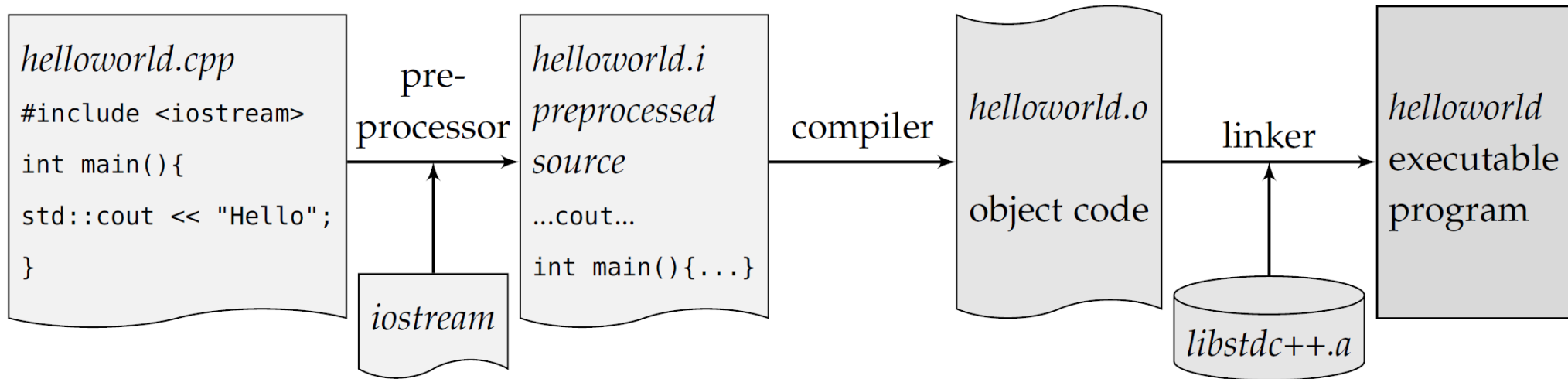
- You can explain the steps required to get from source code to an executable program

- ***.cpp files for source code**

- Also called "Implementation File"
- Function implementations (can be in .h files as well)
- Source of compilation - aka "Translation Unit"

- ***.h or *.hpp files for interfaces (and templates)**

- Also called "Header File"
- Declarations and definitions to be used in other implementation files
- Textual inclusion through a pre-processor (C++20 will incorporate a "Module" mechanism)
- `#include "header.h"`
 - like in C (you know that from Bsys1)



- **C++ is usually compiled into machine code**

- No (Java) Virtual Machine overhead – a bit less flexibility built-in

- **3 Phases of compilation**

- Preprocessor – Textual replacement of preprocessor directives (#include)
- Compiler – Translation of C++ code into machine code (source file to object file)
- Linker – Combination of object files and libraries into libraries and executables

main.cpp

```
#include "sayhello.h"
#include <iostream>

int main() {
    sayHello(std::cout);
}
```

sayhello.h

```
#ifndef SAYHELLO_H_
#define SAYHELLO_H_

#include <iosfwd>

void sayHello(std::ostream&);

#endif /* SAYHELLO_H_ */
```

sayhello.cpp

```
#include "sayhello.h"
#include <ostream>

void sayHello(std::ostream& os) {
    os << "Hi there!\n";
}
```

main.cpp

```
#include "sayhello.h"  
#include <iostream>  
  
int main() {  
    sayHello(std::cout);  
}
```

sayhello.h

```
#ifndef SAYHELLO_H_  
#define SAYHELLO_H_  
  
#include <iosfwd>  
  
void sayHello(std::ostream&);  
  
#endif /* SAYHELLO_H_ */
```



Preprocessor

main.i

```
<content of iosfwd>  
  
void sayHello(std::ostream&);  
  
<content of iostream>  
  
int main() {  
    sayHello(std::cout);  
}
```

main.i

```
<content of iosfwd>

void sayHello(std::ostream&);

<content of iostream>

int main() {
    sayHello(std::cout);
}
```

Compiler

main.o

```
010110101... (machine code)

<Definition of main() which calls
sayHello>
```

main.o

010110101... (machine code)
<Definition of main() which calls
sayHello>

sayhello.o

010110101... (machine code)
<Definition of main() which calls
sayHello>

Linker

sayhello Executable

010110101... (machine code)
<executable program>

C++ Dictionary



Goals:

- You the most frequent basic terminology of C++

Term	Meaning	Example
Value	(Abstract) Element of a type	<code>42</code>
Type	Range of values and behavior	<code>int + - * / %</code>
Variable	Named value holder of a given type	<code>int const answer{42};</code>
Expression	A sequence of operators and operands (e.g. values) to compute a value of a type or to invoke a function	<code>(2 + 4) * 7</code> <code>sayHello(out)</code>
Statement	Executable part of a function body	<code>while (true);</code>
Declaration	Introduction of a new name including its type	<code>void sayHello(std::ostream&);</code>
Definition	Specifies a name and what it is (Every definition is also a declaration)	<code>int size{5};</code>
Function	Executable unit that can be called	<code>void bar() {}</code>

C++ Is NOT Java



Goals:

- You know about the existence of false friends between C++ and Java

- **Beware! Java knowledge may be the wolf in sheep's clothing!**

- Syntax of Java is based on C++'s syntax, which in turn is based on C's syntax

```
public class HelloWorld {  
    public static void main (String[] args) {  
        System.out.println("Hello World");  
    }  
}
```



```
#include <iostream>  
int main() {  
    std::cout << "Hello World\n";  
}
```



```
#include <stdio.h>  
int main() {  
    printf("Hello World\n");  
}
```

Bad Code

- **More C++ vs Java in self-study slides**

- **Java's objects are placed on the heap**

- Exception: Primitive values (int, char, boolean)

```
Type var = new Type();
```




- **C++ allocates memory for variables on definition**

- No **explicit** heap memory needed
- no indirection and space overhead


```
Type var{};
```



```
public class Point {  
    private int x;  
    private int y;  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public static void main(String[] args) {  
        Point point = new Point(1, 20);  
        Point samePoint = point;  
        point.x = 300;  
        System.out.println(samePoint.x);  
    }  
}
```

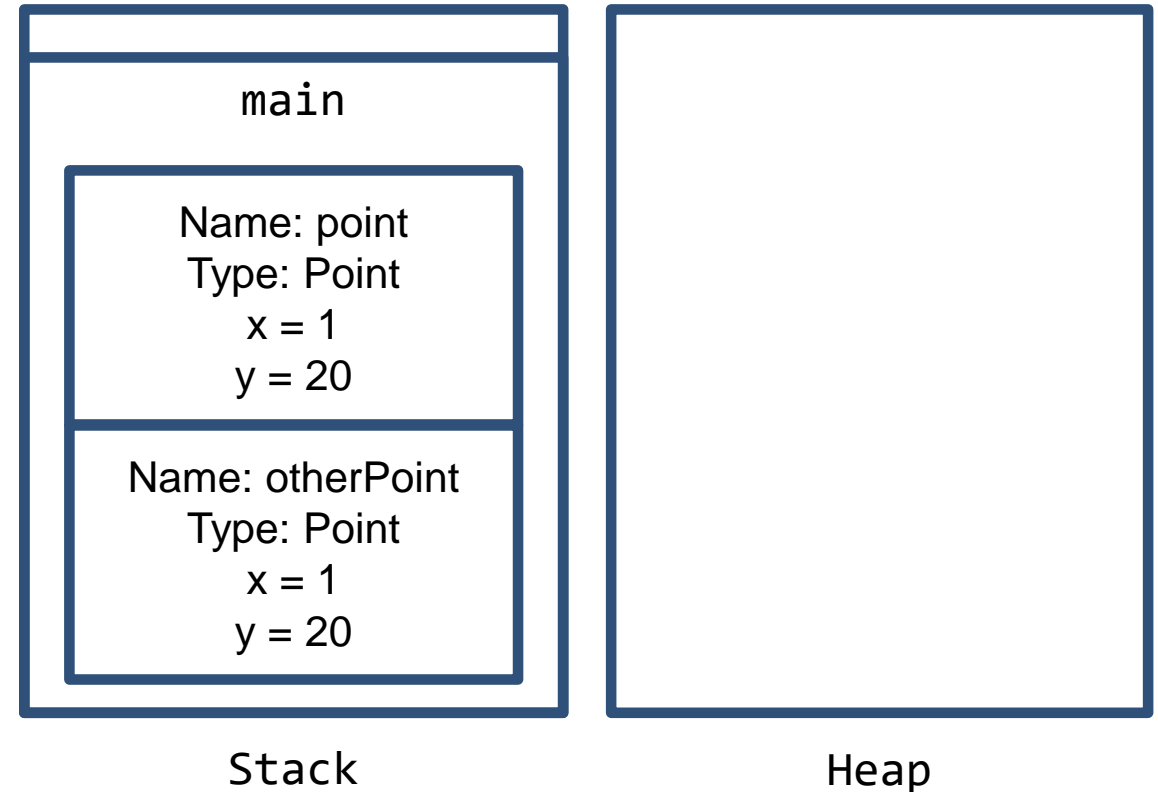
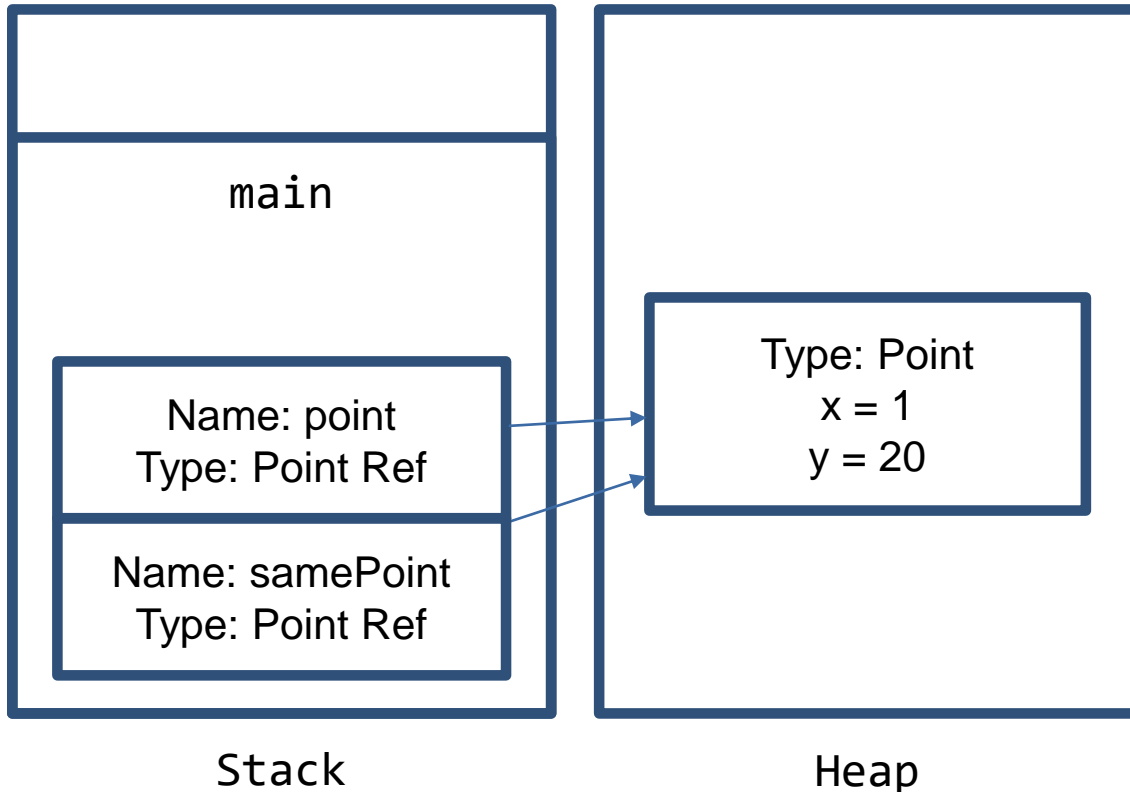


```
#include <iostream>  
  
struct Point {  
    int x;  
    int y;  
};  
  
int main() {  
    Point point{1, 20};  
    Point otherPoint{point};  
    point.x = 300;  
    std::cout << otherPoint.x << '\n';  
}
```

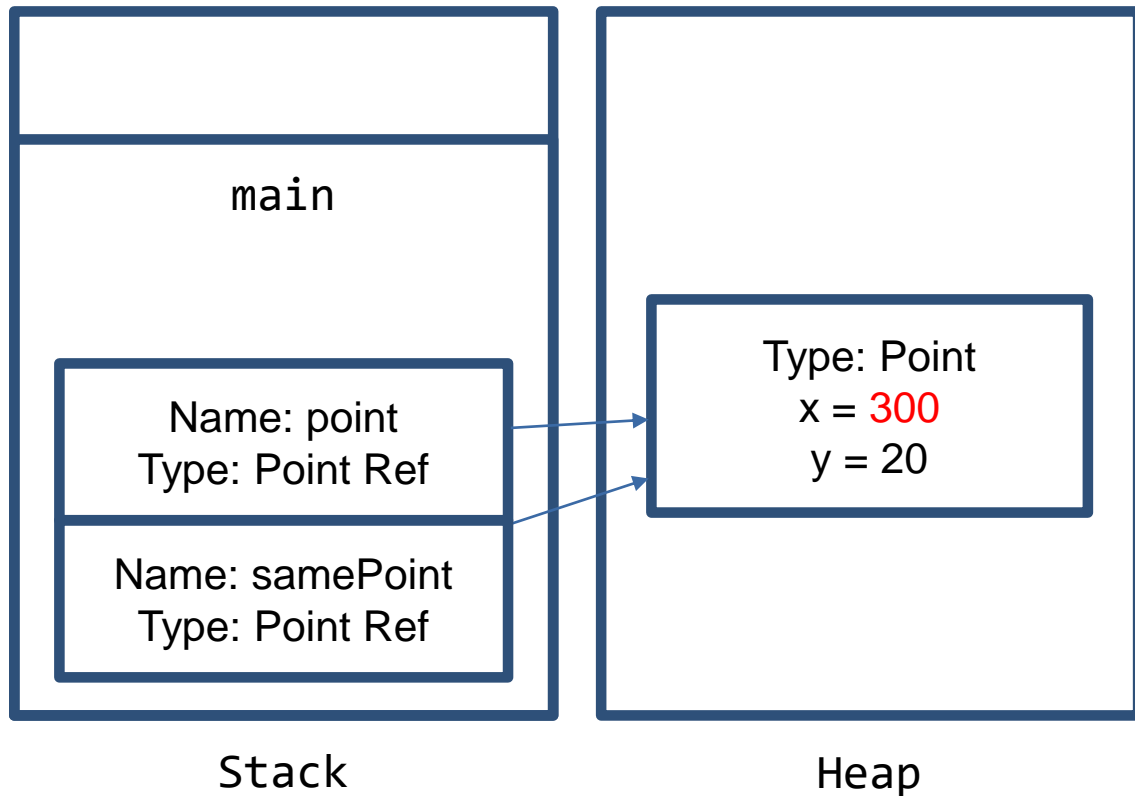


```
public static void main(String[] args) {  
    Point point = new Point(1, 20);  
    Point samePoint = point;  
    point.x = 300;  
    System.out.println(samePoint.x);  
}
```

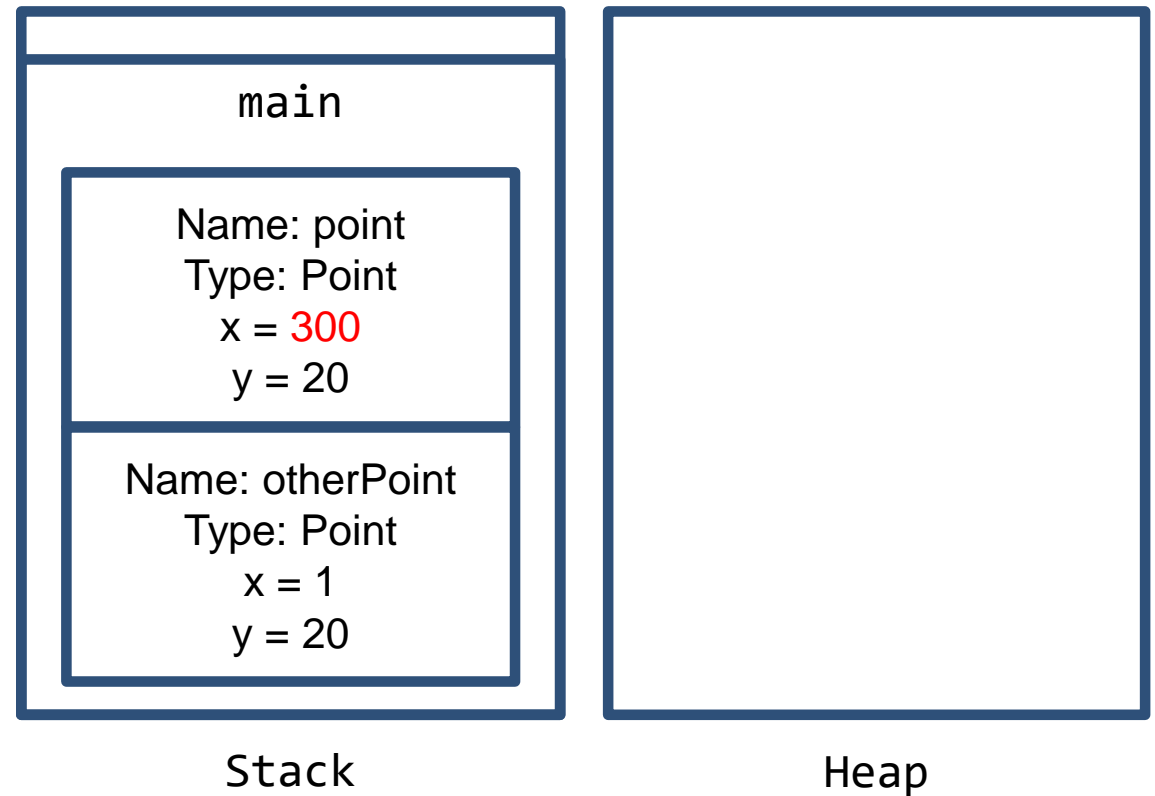
```
int main() {  
    Point point{1, 20};  
    Point otherPoint{point};  
    point.x = 300;  
    std::cout << otherPoint.x << '\n';  
}
```




```
public static void main(String[] args) {  
    Point point = new Point(1, 20);  
    Point samePoint = point;  
    point.x = 300;  
    System.out.println(samePoint.x);  
}
```



```
int main() {  
    Point point{1, 20};  
    Point otherPoint{point};  
    point.x = 300;  
    std::cout << otherPoint.x << '\n';  
}
```



HSR C++Driver's License



Goals:

- You know what we expect from you in this module

- **C++ is a language for professionals**
 - C++11 and later made it easier to learn
 - But added additional power
- **You need to know what happens in your code!**
 - Avoid undefined behavior!
 - Don't be afraid to ask. We gladly help in the exercises.
 - You profit most if you do labs and exercises yourself with others!
- **We will teach beyond syntax and semantics of C++**
 - Software engineering and code quality are as important as correct functionality!



- **Never ever use the following bool expressions/statements in your code, because in the exam you will get points deducted, even if semantically correct!**

```
bool isOdd(int i) {  
    if (i % 2 == 0)  
        return false;  
    else  
        return true;  
}
```

```
bool isEven(int i) {  
    if (i % 2 == 0)  
        return true;  
    else  
        return false;  
}
```

```
bool isNotTrue(bool b) {  
    if (b == false)  
        return true;  
    else  
        return false;  
}
```

- **Better:**

```
bool isOdd(int i) {  
    return i % 2;  
}
```

```
bool isEven(int i) {  
    return !isOdd(i);  
}
```

```
!
```

- **You will need to be able to lookup some details about C++ yourself!**
 - <https://gitlab.dev.ifs.hsr.ch/psommerl/cpp-module>
- **New this semester: Discord for interactive mutual support (<https://discord.gg/vrFXkEJ>)**
 - Please participate actively
- **Online documentation: <https://en.cppreference.com/w/cpp>**
- **Be aware that still a lot of C++ examples on the Internet are bad (and old) code and not C++17!!!!**
- **See <http://isocpp.org> for more information about C++**
 - <https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>
 - C++ Core Guidelines (will be covered also in CPIA) - rules about better/modern and worse/old C++ style
- **If you want to see the compiler in action: <https://godbolt.org>**



- **You are expected that from week 2 on, there do not exist any problems in handling Cevalop, CUTE projects or the Compiler!**
- **You will install and try out Cevalop**
 - If you know how to obtain and install a C++17 capable compiler version
 - We provide Cevalop with our (incomplete) C++17 syntax extensions

(Function) Declarations and Definitions



Goals:

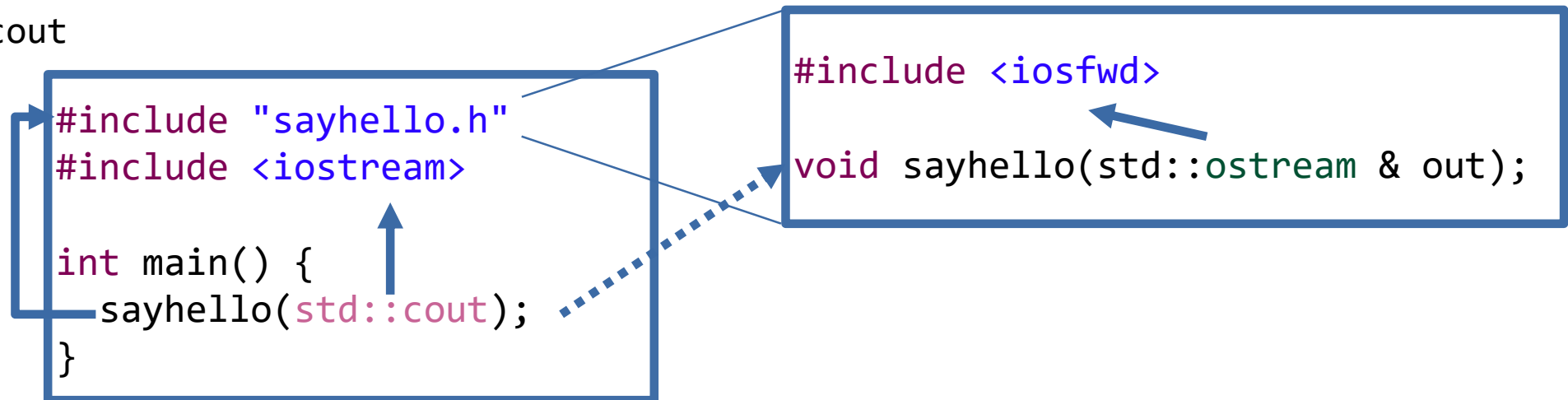
- You know the difference between declarations and definitions
- You can explain the elements of a function definition
- You know about the One Definition Rule

- All things with a name that you use in a C++ program must be declared before you can do so

- E.g. a function that you call (major difference from C)
- A type that you use for a variable (except some built-ins)
- A variable that you use

- For library code such declarations are put in header files

- `#include <iostream>`
- to use `std::cout`



```
<return-type> <function-name>(<parameters>);
```

- **Tells the compiler that there is a function named <function-name> that takes the parameters <parameters> and returns a value of type <return-type>**

Term	Description
Return Type	Every function either returns a value of a specified type or it has return type void
Function Name	Identifier for the function. There can be multiple functions with the same name. Different parameter types are required (Function Overloading)
Parameters	A list of 0 to N parameters. Each parameter has a type and an optional name.
Signature	Combination of name and parameter types. Used for overload resolution (distinction between functions with the same name)

- **Declarations are usually put into a header file (*.h)**
- **There can be multiple declarations of the same function**

```
<return-type> <function-name>(<parameters>) { /*body*/ }
```

- **Implementation of a function**

- Specifies what the function does

Term	Description
Return Type Function Name Parameters Signature	Same as for function declaration
Body	Implementation of the function with 0 to N statements

- **Definitions are usually put into a source file (*.cpp)**
- **There can only be one definition of the same function (One Definition Rule)**

- If a function has a non-void return type it must return a value on every path (or throw an exception)

```
#include "sayhello.h"
#include <ostream>

void sayhello(std::ostream & out) {
    out << "Hello world!\n";
}
```

- main() is special, it has an implicit return statement, returning zero, if not present

```
#include <stdexcept>

int divide(int dividend, int divisor) {
    if (divisor == 0) {
        throw std::invalid_argument{"Divisor must not be 0."};
    }
    return dividend / divisor;
}
```




- While a program element can be declared several times without problem there can be only one definition of it
- This is called the One Definition Rule (ODR)!
- Consequences:
 - There can be only one definition of the `main()` function
 - Or any other function with the same signature
 - There must be a definition for all elements that are used
- **#include guards recommended**
 - not required for function declarations
 - but for (class) type definitions in header files

```
#include "sayhello.h"
#include <iostream>

int main() {
    sayhello(std::cout);
}

int main() {
    saygoodbye(std::cout);
}
```



- Include guards ensure that a header file is only included once
- Multiple inclusions could violate the One Definition Rule when the header contains type definitions

sayhello.h

```
#ifndef SAYHELLO_H_  
#define SAYHELLO_H_  
  
#include <iosfwd>  
void sayhello(std::ostream & out);  
  
#endif /* SAYHELLO_H_ */
```

Be aware to adjust
when copying or
renaming a header file

Directive	Description
#ifndef SYMBOL	Checks whether SYMBOL has already been defined If not the block until #endif is included
#define SYMBOL	Defines SYMBOL
#endif	Closes the block opened by #ifndef

Modularization & Testing



Goals:

- You know how and why to separate functionality into a library
- You setup a Cevelop project with separated unit tests and executable for a library

- **Library functions in separate compilation units**

- Allows unit testing

- Declarations in header files

- **Using the library function requires #include**

- **NOTE: Use "static library project" in Codelab**

- shared libraries can introduce unnecessary hassle

```
#include "sayhello.h"
#include <ostream>
void sayhello(std::ostream & out) {
    out << "Hello world!\n";
}
```

```
#ifndef SAYHELLO_H_
#define SAYHELLO_H_

#include <iosfwd>
void sayhello(std::ostream & out);

#endif /* SAYHELLO_H_ */
```

```
#include "sayhello.h"
#include <iostream>
int main() {
    sayhello(std::cout);
}
```

C++ Library Project

```
#include "sayhello.h"
#include <ostream>
void sayhello(std::ostream & out) {
    out << "Hello world!\n";
}
```

```
#ifndef SAYHELLO_H_
#define SAYHELLO_H_

#include <iosfwd>
void sayhello(std::ostream & out);

#endif /* SAYHELLO_H_ */
```


CUTE Test Project

```
#include "sayhello.h"
#include "cute.h"
#include "ide_listener.h"
#include "cute_runner.h"
#include <sstream>
```

```
void testSayHelloSaysHelloWorld() {
    std::ostringstream out{};
    sayhello(out);
    ASSERT_EQUAL("Hello world!\n", out.str());
}
```

```
void runAllTests() {
    cute::suite s { };
    s.push_back(CUTE(testSayHelloSaysHelloWorld));
    cute::ide_listener<> lis{};
    auto const runner = cute::makeRunner(lis);
    runner(s, "AllTests");
}
```

```
int main() {
    runAllTests();
}
```

- **C++ is substantially different from Java**
 - even though this is not obvious regarding the similarities in the syntax (Java borrowed C++ syntax)
 - Most important difference is the existence of Undefined Behavior 
- **The translation process consists of three main stages (preprocessor, compiler and linker)**
- **C++ function code is separated into declarations (in header files) and definitions (in source files)**
- **The One Definition Rule ensures there is only a single definition for each variable, function and type**
 - violating the ODR can result in Undefined Behavior!
- **Modularization of code enables proper unit testing**